Summary of the OpenSpliceDDS *Tutorial*

Marc Paterno

Fermilab Computing Division
CET group

JDEM SOC Meeting
March 10, 2009

Purpose and scope

- Remind everyone of the purpose of a data distribution system (DDS).
- Report on the highlights of the OpenSpliceDDS tutorial, offered by PrismTech (the company that provides OpenSpliceDDS), and attended by a FNAL colleague and me (and about 10 others).

What is OpenSpliceDDS?

OpenSpliceDDS is

- a publish/subscribe system,
- that implements an Object Management Group (OMG) standard for DDS,
- will “soon” (4–6 weeks, as of 5 March) be open-source software, and
- has a number of “extra value” features, only available in the commercial versions.

PrismTech[1], the company that provides OpenSpliceDDS, say they are migrating their business model toward the “MySQL model”; by this they mean that they'll give away the basic product, and sell both support and “enhanced functionality”.

[1] `http://www.prismtechnologies.com`

What is "standard"

The OMG has standardized

- the language for defining the data types that are published and subscribed to (OMG IDL, the same as is used in CORBA), and for associating names with those types,
- the API for publishing instances of those named types,
- the API for subscribing to named types, or instances of named types,
- the wire protocol used to communicate the data (so that DDS solutions from different vendors can inter-operate).

OpenSpliceDDS implements most of the things that the OMG standardized (the major missing feature is transactions); they also add some things beyond what the standard requires, such as a "more efficient wire protocol". There is no standardization for security; OpenSpliceDDS has an extension that encrypts the data on the wire.

A useful "mental model" for DDS

- One can think of a topic as a database table.
- One can think of an instance of a topic as a row in the database table.
- The DDS system makes the appropriate "data tables" available over the network.
- writers create or modify instances (inserting or updating rows).
- readers are notified when a row changes, so they can (re-)read the row.
- EXCEPT: DDS can keep a history of "samples" for instances, which can be useful.

Supported languages and platforms

OpenSpliceDDS supports, in their open source version,

- C++, Java and C programming languages, and
- Linux and Windows operating systems.

At least one of their customers is working adding support for Python.
They support other operating systems, but only in their commercial versions.
There is currently no support for MacOS.

Personal first impressions

- The presenter of the tutorial was one of the original authors, and was extremely knowledgeable. If this is the caliber of their "support people", their support will be excellent.
- OPENSPLICEDDS provides many "knobs" to tweak in order to optimize system performance.
- Some of these knobs have to do with organizing and controlling use of network resources, *e.g.*, distributing specific topics only to specific multicast addresses, and using domains to partition the physical network into logical segments. It is not clear how, or even if, such things might inter-operate with any "grid technologies"; if the SOC controls its own hardware (rather than relying on shared resources) this may not matter.
- Many of these knobs are in the form of quality of service specifications. Several hours of the tutorial were spent highlighting (!) the ones that are standardized.

- The language APIs are strongly influenced by their OMG IDL specification. This means all APIs are written to a “common subset” of features in the languages supported by OMG IDL.
  - No default values in functions, and no function overloading; explicitly named different functions.
  - No exceptions; functions return error codes, manual checking is necessary.
  - No constructors; objects are made by factories.
  - No covariant return types—explicit casts are needed.
  - No use of C++ standard library, in C++ binding.
- If using this from C++, I’d want to wrap the necessary parts of their API in some modern C++.
- Their Java API involves lots of repetitious boilerplate code: they offer a commercial tool (Eclipse plug-in) that automates much of this generation.

Conclusion

- Our previously-presented plan calls for a 3-month exploration; this still seems reasonable.
- This plan concentrated on measuring performance in a variety of ways. We now know enough about the system to have a good idea of where we want to stress it, and on what performance characteristics we should concentrate.
- My first impression is guardedly optimistic. The product seemed quite mature and full-featured, and worth the time that will be invested in exploring it.